

02

## 기본 자료형과 연산자

# 목차

1. 기본 입력 & 출력 방법
2. 기본 자료형, 변수, 상수
3. 연산자
4. 자료형 변환

# JavaScript 코드 기본

---

## ■ 자바스크립트 코드 위치

- 자바스크립트 코드 위치 : head, body
- head태그 내부의 자바스크립트는 문서 로딩이 완료 안된 상태에서 실행되므로 body태그내의 문서 요소를 참조할 수 없습니다
- head태그 내부의 자바스크립트 코드는 함수 정의, 이벤트 핸들러 정의등이 선언됩니다

# JavaScript 코드 기본

## ■ 표현식과 문장

### ■ 표현식

```
273  
10 + 20 + 30 * 2  
"JavaScript Programming"
```

- 문장 : 표현식이 하나 이상 모일 경우, 마지막에 종결 의미로 세미콜론(;)
- 문장을 구분하는 ;는 가독성을 위해서 사용합니다.(생략가능)

# JavaScript 코드 기본

## ■ 키워드

- 자바스크립트에서 특별한 의미가 부여된 단어

break	else	instanceof	true
case	false	new	try
catch	finally	null	typeof
continue	for	return	var
default	function	switch	void
delete	if	this	while
do	in	throw	with
let	const		

abstract	enum	int	short
boolean	export	interface	static
byte	extends	long	super
char	final	native	synchronized
class	float	package	throws
const	goto	private	transient
debugger	implements	protected	volatile
double	import	public	

# JavaScript 코드 기본

## ■ 식별자

- 이름을 붙일 때 사용하는 단어, 변수와 함수 이름 등으로 사용

- 키워드를 사용 안됨
- 특수 문자는 \_와 \$만 허용
- 숫자로 시작하면 안됨
- 공백은 입력하면 안됨
- 대소문자 구별

alpha	break
alpha10	273alpha
_alpha	has space
\$alpha	
AlPha	
ALPHA	

○ X

# JavaScript 코드 기본

## ■ 식별자 사용 규칙

- 생성자 함수의 이름은 항상 대문자로 시작
- 변수, 함수, 속성, 메소드의 이름은 항상 소문자로 시작
- 여러 단어로 된 식별자는 각 단어의 첫 글자를 대문자로 함

구분	단독으로 사용	다른 식별자와 사용
식별자 뒤에 괄호 없음	변수 또는 상수	속성
식별자 뒤에 괄호 있음	함수	메소드

<code>alert('Hello World')</code>	⇒ 함수
<code>Array.length</code>	⇒ 속성
<code>input</code>	⇒ 변수 또는 상수
<code>prompt('Message', 'Defstr')</code>	⇒ 함수
<code>Math.PI</code>	⇒ 속성
<code>Math.abs(-273)</code>	⇒ 메소드

# JavaScript 코드 기본

## ■ 주석

- 프로그램의 진행에 영향을 주지 않는 코드

방법	표현
한 줄 주석 처리	// 주석
여러 줄 주석 처리	/* 주석 주석 */

```
// 주석은 코드의 실행에 영향을 주지 않습니다.  
/*  
console.log("JavaScript Programming")  
console.log("JavaScript Programming")  
console.log("JavaScript Programming")  
*/
```



# JavaScript 코드 기본

## ■ 출력 메소드

- `console` : 웹 어플리케이션 개발시 디버깅등을 위한 출력 객체
- `log()` : 로깅 함수로 문자열, 숫자, 객체, 배열 등 모든 타입의 데이터를 문자열 형식으로 데이터를 출력할 수 있습니다
- `dir()` : 객체의 속성과 구조를 계층적으로 탐색하여 출력합니다.

```
console.log(message, ...optionalParams)
```

```
const user = { name: 'Alice', age: 25 };  
console.log('Hello, world!');  
console.log('User:', user);  
console.log('Age:', user.age);
```

```
console.dir(obj, [options])
```

```
console.dir(window);  
console.dir(document);
```

# JavaScript 코드 기본

## ■ 출력 메소드

- `document.write()` : HTML 문서의 콘텐츠를 변경하는 데 사용됩니다
- `window.alert()` : 경고 대화 상자를 표시합니다.

```
document.write(content)
```

```
<body>  
  <script>  
    document.write('<h1>Hello, World!</h1>');  
    document.write('<p>This is a paragraph.</p>');  
  </script>  
</body>
```

```
window.alert(message)
```

```
<script>  
  window.alert('Hello, this is an alert!');  
</script>
```

# JavaScript 코드 기본

## ■ 이스케이프 문자

- 문자열 내에서 특별한 의미를 갖는 문자나 특수 문자를 포함하기 위해 이스케이프 문자를 사용합니다.
- 백슬래시(\)로 시작

Escape문자	설명
\n	새 줄(줄 바꿈)을 나타냅니다. 문자열을 여러 줄로 나누고자 할 때 사용
\t	탭 문자를 나타냅니다. 문자열 내에 탭 공간을 삽입할 때 사용
\\	백슬래시 자체를 나타냅니다. 백슬래시를 문자열에 포함시키기 위해 사용
\"	큰따옴표를 나타냅니다. 큰따옴표를 포함한 문자열을 정의할 때 사용
\'	작은따옴표를 나타냅니다. 작은따옴표를 포함한 문자열을 정의할 때 사용
\r	캐리지 리턴을 나타냅니다. 주로 새 줄과 함께 사용되며, 텍스트의 줄의 시작으로 커서를 이동시킵니다.
\b	백스페이스 문자를 나타냅니다. 이전 문자를 삭제합니다.

```
console.log("이름\t나이");  
console.log("안녕\n하세요");  
console.log("\\\\");
```

# JavaScript 코드 기본

## ■ 입력 메소드

- `prompt()` : 사용자에게 텍스트 입력을 요청하는 대화 상자를 표시합니다.
- `confirm()` : 사용자가 특정 작업을 확인하거나 취소할 수 있는 대화 상자를 표시합니다

```
prompt(message, default)
```

```
<body>  
  <script>  
    let name = prompt('What is your name?', 'Guest');  
  </script>  
</body>
```

```
window.alert(message)
```

```
<script>  
  let result = confirm('Do you really want to delete this item?');  
</script>
```

# JavaScript 코드 기본

## ■ 입력

- 웹 브라우저에서 작동하는 자바스크립트 : `prompt( )` 이름의 함수를 받음
- `readline` 모듈 : 터미널에서 사용자 입력을 쉽게 처리할 수 있도록 도와주는 모듈
- `readline.createInterface()`를 사용하여 input, output을 설정
- `rl.question()`을 사용하여 단일 입력을 받을 수 있음
- `rl.on("line")` 이벤트를 활용하여 여러 줄 입력 가능

```
// 표준 입력을 다루기 위한 readline 모듈 가져오기
const readline = require("readline");

// 인터페이스 생성
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

// 사용자 입력 받기
rl.question("당신의 이름은 무엇인가요? ", (answer) => {
  console.log(`안녕하세요, ${answer}!`);
  rl.close(); // 입력 종료
});
```

# JavaScript 코드 기본

## ■ 입력

- process.stdin 객체 사용, 이벤트 리스너("data", "end", "close" 등)를 사용하여 입력 처리
- setEncoding("utf8")로 입력을 문자열로 변환 가능
- process.exit()을 사용하여 프로그램 종료 가능

```
// 여러 줄 입력 받기 (line 이벤트 사용)
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

console.log("이름을 입력하세요. 종료하려면 'exit'를 입력하세요.");

rl.on("line", (input) => {
  if (input === "exit") {
    console.log("입력을 종료합니다.");
    rl.close();
  } else {
    console.log(`입력한 이름: ${input}`);
  }
});
```

# 기본 자료형

## ■ 기본 자료형

자료형	설명
string	텍스트 데이터를 저장하고 조작하는 데 사용됩니다. 작은따옴표(""), 큰따옴표(""), 또는 백틱(`)으로 감쌀 수 있습니다.
number	정수 및 부동 소수점 숫자를 포함하는 숫자를 나타내는 자료형
boolean	논리적 참(true) 또는 거짓(false)을 나타내는 자료형
object	키-값 쌍의 컬렉션을 나타내는 복합 자료형 여러 속성 및 메서드를 가질 수 있습니다.
function	코드 블록을 캡슐화하여 나중에 실행할 수 있는 특별한 자료형 함수는 일급 객체로, 변수에 할당하거나 다른 함수에 인수로 전달할 수 있습니다
undefined	값이 할당되지 않은 변수를 나타냅니다. 변수를 선언했으나 초기화하지 않은 자료형
null	의도적으로 "값 없음"을 표현
BigInt	범위 제한 없는 큰 정수 (예: 123456789012345678901234567890n)
Symbol	고유한 값

# 기본 자료형

## ■ 기본 자료형

```
const sym1 = Symbol("id"); // 새로운 Symbol 생성
const sym2 = Symbol("id");

console.log(sym1 === sym2); // false (고유한 값이므로 서로 다름)

const user = {
  name: "Alice",
  [sym1]: 1234, // Symbol을 객체 키로 사용
};

console.log(user.name); // "Alice"
console.log(user[sym1]); // 1234

// Symbol 속성은 Object.keys(), for...in에서는 출력되지 않는다 (숨겨진 속성)
for (let key in user) {
  console.log(key);
}

console.log(Object.keys(user)); // ["name"]
console.log(Object.getPrototypeOfSymbols(user)); // [ Symbol(id) ]
```



# 기본 자료형

## ■ 기본 자료형

```
// BigInt 는 n을 붙이거나 BigInt()로 생성 가능
const big1 = 1234567890123456789012345678901234567890n;
const big2 = BigInt("9876543210987654321098765432109876543210");

console.log(big1); // 1234567890123456789012345678901234567890n
console.log(big2); // 9876543210987654321098765432109876543210n

// Number 타입과 직접 산술연산 불가 (TypeError 발생)
// console.log(big1 + 10); // TypeError

// BigInt끼리 연산 가능
const sum = big1 + big2;
console.log(sum); // 11111111101111111111011111111011111111100n

// Number 타입과 비교 연산은 가능
console.log(big1 > 1000); // true

// Number로 변환 (정밀도 손실 가능)
console.log(Number(big1)); // 1.2345678901234568e+39
```

# 기본 자료형

## ● null & undefined

undefined	null
변수가 선언되었지만 값이 할당되지 않은 상태	null은 의도적으로 "값이 없음"을 나타내기 위해 할당된 값
변수 선언 후 값을 초기화하지 않으면 자동으로 undefined가 할당됨	개발자가 특정 변수나 속성에 값이 없음을 명시적으로 표현하고자 할 때 사용
객체의 속성에 접근하려 했는데 해당 속성이 존재하지 않으면 undefined를 반환	null은 프로그래머가 수동으로 값을 설정해야 합니다.
함수가 명시적으로 값을 반환하지 않으면 기본적으로 undefined를 반환	객체가 없거나 유효하지 않은 상태를 나타낼 때 사용됨
typeof null은 object를 반환합니다.	
동등 연산자(==): undefined와 null을 동일하게 취급합니다.	
일치 연산자(===): undefined와 null은 엄격히 다른 값으로 간주합니다.	

# 자료형 검사

- 자료형 확인 연산자

자료형 확인 연산자

연산자	설명
typeof	해당 변수의 자료형을 추출합니다.

# 강제 자료형 변환

## 강제 자료형 변환 함수

함수	설명
Number()	숫자로 자료형 변환합니다.
String()	문자열로 자료형 변환합니다.
Boolean()	불로 자료형 변환합니다.

### ■ Number( ) 함수와 NaN

```
console.log(Number("52"));  
console.log(Number("52.273"));  
console.log(Number(true));  
console.log(Number(false));  
console.log(Number("안녕하세요"));
```

# 변수 & Scope

- 과거 자바스크립트에서는 var 키워드를 사용했으나 스코프 문제가 많아 현재는 let 키워드를 사용함

var 변수	let 변수
같은 이름의 변수를 <b>중복 선언</b> 할 수 있으며, <b>재할당</b> 도 가능합니다.	동일 스코프 내에서 중복 선언 불가
<b>함수 스코프</b> 를 따르며, 블록 스코프를 무시합니다. 블록 {} 내에서 선언된 var 변수는 블록 외부에서도 접근 가능	<b>블록 스코프</b> 를 따릅니다.
<b>변수 호이스팅</b> 이 발생하여 변수 선언이 해당 스코프의 최상위로 끌어올려집니다. 할당은 원래 위치에서 이루어집니다.	
초기값이 할당되기 전에 접근하면 'undefined'가 반환됩니다	
<b>전역 객체</b> (window 객체 또는 global 객체)의 속성이 됩니다.	let으로 선언된 전역 변수는 전역 객체의 속성이 되지 않습니다.

```
{  
  let a = 10;  
}  
  
console.log(a);
```

```
{  
  var a = 10;  
}  
  
console.log(a);
```

# 변수 & Scope

## ● 스코프 Scope

- 변수를 사용할 수 있는 범위
- 스코프 == 블록
- 블록 내부에 선언된 변수는 해당 블록 내부에서만 사용가능
- 반복문에 선언된 변수는 반복문 블록 내부에서만 활용, 외부에서 활용 불가

```
{  
    let a = 10;  
}  
  
console.log(a);
```

```
for (let i = 0; i < 3; i++) {  
    console.log(i);  
}  
  
console.log(i);
```

변수 i는 해당블록 외부에서  
사용할 수 없습니다.

# 변수 & Scope

---

- 호이스팅 Hoisting

- 블록에서 사용할 변수를 미리 확인해서 실행 전에 생성하는 작업

```
let a = 1;  
{  
  console.log(a);  
  let a = 2;  
}
```

# 변수 & Scope

---

## ■ 상수

- 상수 : '항상 같은 수'라는 의미, 변수와 반대되는 개념
- `const` : 상수constant를 만드는 키워드
- 변하지 않을 대상에 상수를 적용

```
// 상수를 선언합니다.  
const constant = "변경할 수 없어요";  
constant = "";  
  
// 출력합니다.  
console.log(constant);
```



# 연산자

## ■ 문자열 연산자

연산자	설명
문자열[숫자]	문자열 선택 연산자
+	문자열 연결 연산자

```
console.log("안녕하세요"[0]);  
console.log("안녕하세요"[1]);  
console.log("안녕하세요"[3]);
```

```
> `올해는 ${new Date().getFullYear()}년입니다.`
```

# 연산자

## ■ 문자열 연산자

연산자	설명
**	거듭제곱 연산자, Math.pow() 의 결과가 동일

```
console.log(2 ** 3);  
console.log(5 ** 2);
```

```
console.log(Math.pow(2, 3)); // // 기존 Math.pow()와 비교
```

# 연산자

## ■ 비교 연산자

연산자	설명
==	같습니다.
!=	다릅니다.
>	왼쪽 피연산자가 큼니다.
<	오른쪽 피연산자가 큼니다.
>=	왼쪽 피연산자가 크거나 같습니다.
<=	오른쪽 피연산자가 크거나 같습니다.

연산자	설명
===	자료형과 값이 같은지 비교합니다.
!==	자료형과 값이 다른지 비교합니다.

```
console.log(`52 == "52": ${52 == "52"}`);  
console.log(`52 === "52": ${52 === "52"}`);  
console.log();  
console.log(`${0 == ""}: ${0 == ""}`);  
console.log(`${0 === ""}: ${0 === ""}`);
```

# 연산자

## ■ 증감 연산자

연산자	설명
변수++	기존 변수 값에 1을 더합니다(후위).
++변수	기존 변수 값에 1을 더합니다(전위).
변수--	기존 변수 값에서 1을 뺍니다(후위).
--변수	기존 변수 값에서 1을 뺍니다(전위).

# 연산자

---

## ■ 논리 연산자

연산자	설명
!	논리 부정 연산자
	논리합 연산자
&&	논리곱 연산자

# 연산자

## ■ 대입 연산자

숫자에 적용하는 복합 대입 연산자

연산자	설명
<code>+=</code>	숫자 덧셈 후 대입 연산자
<code>-=</code>	숫자 뺄셈 후 대입 연산자
<code>*=</code>	숫자 곱셈 후 대입 연산자
<code>/=</code>	숫자 나눗셈 후 대입 연산자

문자열에 적용하는 복합 대입 연산자

연산자	설명
<code>+=</code>	문자열 연결 후 대입 연산자

# 연산자

## ■ 클래스와 인스턴스의 비교 연산자

연산자	설명
instanceof	객체가 특정 클래스(또는 생성자 함수)의 인스턴스인지 확인
Symbol.hasInstance	instanceof의 동작을 커스텀 로직으로 만들 수 있음

```
class Animal {}
class Dog extends Animal {}
const dog = new Dog();

console.log(dog instanceof Dog); // true
console.log(dog instanceof Animal); // true
console.log(dog instanceof Object); // true
console.log(dog instanceof Array); // false

class CustomType {
  static [Symbol.hasInstance](instance) {
    return instance.canRun === true;
  }
}

const obj = { canRun: true };
console.log(obj instanceof CustomType); // true (기본적인 instanceof 동작을 변경)
```

# 연산자

## ■ 널 병합 , 옵셔널 체이닝 연산자

연산자	설명
??	null 또는 undefined일 때만 기본값을 설정하는 연산자, ES11
?.	객체 속성이 undefined거나 null이어도 안전하게 접근 가능, ES11

```
let x = null ?? "default";  
console.log(x); // "default"
```

```
let y = 0 ?? "default";  
console.log(y); // 0 (0은 null이나 undefined가 아니므로 유지됨)
```

```
const user = { name: "Alice" };  
console.log(user?.name); // "Alice"  
console.log(user?.address?.city); // undefined (에러 발생하지 않음)
```



# 연산자

## ■ 논리 할당 연산자

연산자	설명
&&=,   =, ??=	값이 특정 조건을 만족할 때만 할당

```
let a = true;  
a &&= "Hello"; // a가 true이면 "Hello"로 변경  
console.log(a); // "Hello"
```

```
let b = false;  
b ||= "World"; // b가 falsy이면 "World"로 변경  
console.log(b); // "World"
```

```
let c = null;  
c ??= "Fallback"; // c가 null 또는 undefined이면 "Fallback"으로 변경  
console.log(c); // "Fallback"
```

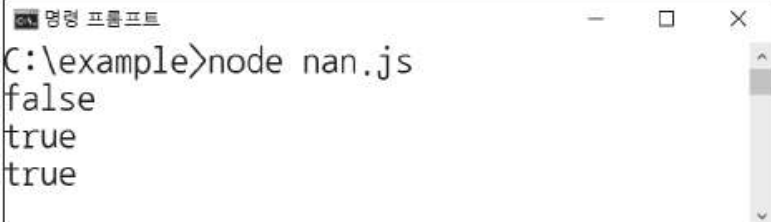
# 자료형 변환

- '숫자로 변환할 수 없는 문자열'을 Number( ) 함수로 변환하면 'NaN'을 출력
- NaN(Not a Number)은 '숫자 자료형이지만 숫자가 아닌 것'을 의미
- NaN의 특징
  - NaN과 NaN의 비교연산은 false
  - NaN인지 확인할 때는 isNaN( ) 함수를 사용

코드 2-24 Not a Number

nan.js

```
// NaN 변수를 만듭니다.  
let nan = Number("안녕하세요");  
  
// NaN끼리 비교합니다.  
console.log(nan == nan);  
console.log(nan != nan);  
  
// isNaN() 함수로 NaN인지 확인합니다.  
console.log(isNaN(nan));
```

명령 프롬프트 창은 'C:\example>node nan.js' 명령을 실행한 후 세 줄의 출력을 보여줍니다. 첫 줄은 'false'로, NaN과 NaN의 등호 비교 결과가 거짓임을 나타냅니다. 두 번째 줄은 'true'로, NaN과 NaN의 부등호 비교 결과가 참임을 나타냅니다. 세 번째 줄은 'true'로, isNaN 함수가 NaN 값을 올바르게 식별했음을 나타냅니다.

```
C:\example>node nan.js  
false  
true  
true
```

# 자료형 변환

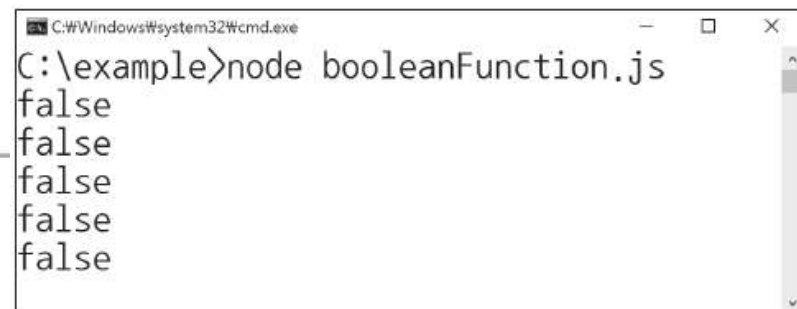
## ■ Boolean( ) 함수

- Boolean( ) 함수를 사용하면 5개의 요소는 false로 변환
- 0, NaN, ""[빈 문자열], null, undefined

코드 2-25 Boolean() 함수

booleanFunction.js

```
// 변수를 선언합니다.  
let nan = Number("안녕하세요");  
let undefinedVariable;  
  
// Boolean() 함수를 사용합니다.  
console.log(Boolean(0));  
console.log(Boolean(nan));  
console.log(Boolean(""));  
console.log(Boolean(null));  
console.log(Boolean(undefinedVariable));
```



A terminal window titled 'C:\Windows\system32\cmd.exe' showing the command 'C:\example>node booleanFunction.js' and its output, which consists of five 'false' values, one for each line of the JavaScript code shown in the previous block.

```
C:\Windows\system32\cmd.exe  
C:\example>node booleanFunction.js  
false  
false  
false  
false  
false
```

# 자료형 변환

---

## ■ 숫자와 문자열 자료형 자동 변환

- 숫자와 문자열에 '+' 연산자를 적용하면 자동으로 숫자가 문자열로 변환

```
console.log(52 + 273);  
console.log("52" + 273);  
console.log(52 + "273");  
console.log("52" + "273");
```

# 자료형 변환

---

## ■ 논리 자료형의 형 변환

```
// 변수를 선언합니다.  
let nan = Number("안녕하세요");  
let undefinedVariable;  
  
// 부정 연산자를 두 번 사용합니다.  
console.log(!!0);  
console.log(!!nan);  
console.log(!!"");  
console.log(!!null);  
console.log(!!undefinedVariable);
```